EXHIBIT 4-3

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, vs. GOOGLE INC. Defendants.	Case No. 3:10-cv-03561-WHA

EXPERT REPORT OF DR. BENJAMIN F. GOLDBERG REGARDING VALIDITY OF PATENTS-IN-SUIT SUBMITTED ON BEHALF OF PLAINTIFF ORACLE AMERICA, INC.

pa-1475723

I, Benjamin F. Goldberg, Ph.D., submit the following expert report ("Patent Validity Report") on behalf of plaintiff Oracle America, Inc. ("Oracle"):

I. INTRODUCTION

1. This report pertains to my review, analysis, and opinions regarding the validity of the patents-in-suit asserted by Oracle against Google in the case known as *Oracle America, Inc. v. Google, Inc.*, pending in the United States District Court for the Northern District of California, Case No. 10-03561-WHA.

2. Of note, my report contains a detailed rebuttal to the seven validity reports submitted by four different Google experts. My failure to address any specific statement in any of Google's reports does not imply that I agree with the statement, and no such agreement should be inferred.

A. Retention

3. I have been retained to consult with Oracle's counsel, review documents and other information, prepare expert reports, and be available to testify regarding my opinions on behalf of Oracle in connection with litigation brought by Oracle against Google.

4. I have been asked to render opinions concerning the validity of the asserted claims of the patents-in-suit in light of the prior art and other validity challenges put forth by Google discussed in this report.

B. Scope of Work Performed and Expected Testimony

5. In general I have reviewed materials to provide technical teaching and opinions regarding the validity of the patents-in-suit.

6. As stated in Oracle's list of issues for expert testimony, I expect to testify at trial regarding:

a. State of the art, subject matter, novelty, and significance of the claimed inventions in relation to the patents-in-suit;

b. Priority dates of the patents-in-suit;

execute the indicated action. It is inefficient, however, for interpreters to resolve the same symbolic references repeatedly. As James Gosling, the inventor of the '104 patent, explained, "each time an instruction comprising a symbolic reference is interpreted, execution is slowed significantly." ('104, 2:13-15.) Accordingly, there was a long-felt need to increase the speed at which interpreters executed code containing symbolic references.

421. The '104 patent satisfied this need by designing an interpreter that operated on intermediate form object code and, whenever it resolves a symbolic reference to data, stores the corresponding numerical (*i.e.*, memory location) reference for later use. (*See generally* '104 patent.) When the interpreter described in the patent encounters a subsequent reference to the data, it simply goes to the corresponding memory location rather than performing another time-consuming symbolic reference resolution. (*See, e.g.*, *id.* at 2:35-59.) The '104 patent thus eliminated the need to resolve the same symbolic reference twice. (*See, e.g.*, '104, 2:60-67.) As summarized in the '104 patent:

> As a result, the 'compiled' intermediate form object code of a program achieves execution performance substantially similar to that of the traditional compiled object code, and yet it has the flexibility of not having to be recompiled when the data objects it deals with are altered like that of the traditional translated code, since data reference resolution is performed at the first execution of a generated instruction comprising a data reference. (*Id.* at 2:60-67.)

422. The '104 patent reduced the number of symbolic reference resolutions that occur during run time and thus solved the need to quickly execute intermediate form object code having symbolic references.

2. The '104 Patent Led to Commercial Success

423. I understand that Sun Microsystems and Oracle have implemented the claimed invention of the '104 patent in their Java virtual machines. In May 1996, James Gosling and Henry McGilton co-authored a white paper entitled "The Java Language Environment," in which they describe symbolic reference resolution for Java. (James Gosling & Henry McGilton, *White*

Paper, The Java Language Environment (May 1996), *available at* http://java.sun.com/docs/white/langenv/.) The white paper documents the core pieces of Java, including symbolic reference resolution as disclosed in the '104 patent.

424. The white paper explains, "Java's memory management model is based on *objects* and *references* to objects." (*Id.* at ch.2.1.6 (emphases in original).) Java bytecode references objects via symbolic references "that are resolved to real memory addresses at run time by the Java interpreter." (*Id.* at ch.6.1.) The chapter on "Interpreted and Dynamic" further explains symbolic reference resolution:

> The Java compiler doesn't compile references down to numeric values—instead, it passes symbolic reference information through to the byte code verifier and the interpreter. The Java interpreter performs final name resolution once, when classes are being linked. Once the name is resolved, the reference is rewritten as a numeric offset, enabling the Java interpreter to run at full speed. (*Id.* at ch.5.1.2.)

425. Therefore, Java interpreter only needs to incur "the small expense of a name lookup the first time any name is encountered" and need not incur the expense the second time that name is encountered. (*Id.*) After the interpreter performs the first name lookup, it can simply reference the "numeric offset." (*Id.*) In this way, the '104 patent allows "the Java interpreter to run at full speed." (*Id.*)

426. Others in the field have recognized Java's execution performance. (*See, e.g.*, Patrick Niemeyer & Joshua Peck, *Exploring Java*, Ch. 1.2 (O'Reilly 2d Ed. 1997), *available at* http://doc.novsu.ac.ru/oreilly/java/exp/index.htm (Although "[i]n general, interpreters are slow . . . Java is a fast interpreted language.").) Some consider Java as "a top performer along with C++ in many cases" even though Java requires an extra step of interpretation. (Carmine Mangione, *Performance tests show Java as fast as C++*, JavaWorld (Feb. 1, 1998), *available at* http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf.html.)

427. I understand that testimony at trial will show customer demand for devices with faster execution performance. Because the '104 patent increases Java interpreters' execution

speed, it has contributed to Java's acceptance in the market as a fast interpreted language. Therefore, I understand that there is a nexus between the claimed invention of the '104 patent and Java's commercial success.

428. Similarly, the '104 patent helps Android achieve good execution performance. I have read Professor Mitchell's Opening Patent Infringement Report, Section VI entitled "RE38,104 (Reference Resolution)" and understand that the evidence at trial will show that Android's Dalvik VM and the dexopt tool that optimizes .dex files both employ the '104 patent. Specifically, I understand that the evidence at trial will show that Dalvik VM and dexopt replace symbolic references with numeric references such as a simple integer v-table offset. Google has characterized the symbolic reference resolution as an "optimization" and has featured it in a presentation describing the implementation of the Dalvik virtual machine. (Google I/O Android Video on "Dalvik Virtual Machine Internals" by Dan Bornstein (2008), *available at* http://developer.android.com/videos/index.html#v=ptjedOZEXPM.) Therefore, Google also acknowledges how symbolic reference resolution increases execution speed and has marketed its significance through a Google I/O presentation to software developers.

429. Furthermore, I have read Professor Mitchell's Opening Patent Infringement Report, Section IV B, entitled "The Claimed Invention in the Patents-in-Suit are Necessary to Achieve Sufficient Performance and Security". I understand that Dr. Mitchell, in consultation with Oracle Java engineers Bob Vandette and Dr. Peter B. Kessler, have conducted benchmark testing and analysis of the technology described in the '104 patent, and they have confirmed that the performance of Android would be poor without the benefit of using the '104 patent. I further understand that the performance benchmark testing results show an execution speed improvement of as much as 13 times with the '104 patent than without the '104 patent.

430. I understand that testimony at trial will show customer demand for devices with faster execution performance. Based on the benchmark analysis, I conclude that Android would have been a slower, and thus less attractive platform if it had not implemented the '104 patent.

Therefore, I understand that there is a nexus between the claimed invention of the '104 patent and Android's commercial success.

431. For at least the above reasons, it is my opinion that secondary considerations demonstrate the non-obviousness of the '104 patent.

B. '205 patent

1. The '205 Patent Solved a Long-Felt Need

432. Traditional Just-In-Time ("JIT") Java compilers translate Java bytecode into native machine code continuously during runtime, compiling the bytecode "just-in-time" before it is about to be loaded or executed. Traditional JIT compilers then cache the compiled code for later use. Symantec Corporation's JIT compiler, which Sun licensed and integrated into JDK 1.1, is an example of such a traditional JIT compiler. (*See* Symantec's Just-In-Time Java Compiler to be Integrated Into Sun JDK 1.1 (Apr. 7, 1997), *available at* http://www.symantec.com/about/news/release/article.jsp?prid=19970407_03 (Symantec's JIT compiler "instantly convert[s] Java bytecode to native code on the fly . . . .").)

433. With JIT compilation, "[O]verall program execution time now includes JIT compilation time, in contrast to the traditional methodology of performance measurement, in which compilation time is ignored." (Ali-Reza Adl-Tabatabai et al., *Fast, Effective Code Generation in a Just-In-Time Java Compiler*, 33 PLDI '98 Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design & Implementation, 280, 280 (1998).) "As a result, it is extremely important for the compiler optimizations to be lightweight and effective." (*Id.*) Furthermore, "native code generated by a JIT compiler does not always run faster than code executed by an interpreter. For example, if the interpreter is not spending the majority of its time decoding the Java virtual machine instructions, then compiling the instructions with a JIT compiler may not increase the execution speed." ('205, 2:5-10.) Accordingly, as the '205 patent inventors recognized, "there [was] a need for new techniques for increasing the execution speed of computer programs that are being interpreted." (*Id.* at 2:27-29.) "Additionally, there [was] a

need for new techniques that provide flexibility in the way in which interpreted computer programs are executed." (*Id.* at 2:29-31.)

434. The '205 patent satisfied this long-felt need by providing a hybrid code execution technique that combines interpretation and execution of compiled code. The claimed invention compiles a *portion* of Java bytecode into native machine code and has a mechanism to switch between execution of Java bytecode and compiled native machine code during runtime. The claimed invention can choose to compile and optimize only the frequently-executed bytecode into native machine code. It can forego JIT compilation of less-frequently-executed bytecode, thereby reducing overall compilation time and memory usage (because compiled code typically takes up more space than bytecode). The claimed invention thus increases execution speed of Java bytecode in a flexible way.

2. The '205 Patent Led to Commercial Success

435. I understand that the HotSpot virtual machine first became an optional component of JDK 1.2 on or about March 18, 1998. (*See* Plaintiff Oracle America, Inc.'s Objections & Responses to Defendant Google, Inc.'s Fourth Set of Interrogatories at 3.)

436. I understand that Java engineer Dr. Peter Kessler has confirmed that the HotSpot virtual machine employs the _fast_invokevfinal() nonstandard bytecode to implement the asserted claims of the '205 patent. (8/4/2011 Kessler Dep. 53:16-55:10, 63:2-14, 64:15-68:13.) I also understand that Dr. Kessler testified that JDK 1.2 and subsequent versions, and HotSpot 1.0 and subsequent versions practice the asserted claims of the '205 patent. (*Id.* at 179:4-11.)

437. The _fast_invokevfinal() bytecode is implemented in the OpenJDK 7 version of HotSpot. (*See* Bytecodes.java, *available at* http://www.docjar.com/html/api/sun/jvm/hotspot/interpreter/Bytecodes.java.html.) The _fast_invokevfinal() method is also implemented in the JDK 6 version of HotSpot. *See* Bytecodes.cpp for JDK 6, *available at* http://openjdk.java.net/projects/jdk6/ (JDK-6u21|hotspot\src\share\vm\interpreter).)

files and places the constants in a separate shared table. ('702, 5:6-17.) Because the resulting multi-class file – comprising the reduced class files and the shared table – contains only one copy of constants and the reduced class files do not contain the duplicated constants, the patented approach significantly reduces the memory usage.

2. The '702 Patent Led to Commercial Success

454. The '702 patent has contributed to Android's commercial success. I have read Professor Mitchell's Opening Patent Infringement Report, Section VIII entitled "5,966,702 (Class File Redundancy Removal", and understand that the evidence at trial will show that Android's dx tool implements the '702 patent. Specifically, I understand that the evidence at trial will show that the dx tool pre-processes class files into a .dex file format that can be interpreted by the Dalvik Virtual Machine. I also understand that the evidence at trial will show that the dx tool pre-processes class files by removing the duplicated constants from multiple class files and storing them in a shared constant pool in a .dex file.

455. Google has presented and therefore marketed how the shared constant pool minimizes memory space. According to a Google I/O presentation, Google has "collapsed all of those, the separate constant pools into one constant pool . . . . [T]here's less total storage required for this shared constant pool." (Google I/O Android Video on "Dalvik Virtual Machine Internals" by Dan Bornstein (2008), *available at* http://developer.android.com/videos/index.html#v=ptjedOZEXPM.) Based on Professor Mitchell's Opening Patent Infringement Report, I understand that the excerpt above refers to the functionalities of the '702 patent.

456. Furthermore, I have read Professor Mitchell's Opening Patent Infringement Report, Section IV entitled "Android's Success Is Due to the Claimed Inventions" and understand that Dr. Mitchell, together with Oracle Java engineer Noel Poore, have conducted benchmark testing and analysis, which has confirmed that the performance of Android would be poor without the benefit of using the '702 patent. I further understand that the performance

benchmark testing results show that a .dex file for an application is between 1.45 and 3.33 times smaller than it would be if duplicate constant removal were not performed.

457. I understand that testimony at trial will show customer demand for devices that use memory efficiently and thus can store more applications or data. Based on the benchmark analysis, I conclude that Android would not have same the capacity to store and run large numbers of applications as it currently has if it had not implemented the '702 patent. I conclude that Android would be a less attractive platform if it had not implemented the '702 patent. Therefore, I understand that there is a nexus between the claimed invention of the '702 patent and Android's commercial success.

458. For at least the above reasons, it is my opinion that secondary considerations demonstrate the non-obviousness of the '702 patent.

D. '520 patent

1. The '520 Patent Solved a Long-Felt Need

459. Conventional Java compilers generate a method called <clinit> to perform class initialization, including initialization of static arrays. ('520, 1:57-62.) The inventors of the '520 patent recognized that, when using bytecode methods for array initialization, "the amount of code required to initialize the array is many times the size of the array, thus requiring a significant amount of memory." (*Id.* at 2:53-58.) Given the limited amount of memory available on devices, and particularly on embedded devices, there was a need to initialize arrays using less memory space.

460. The '520 patent solved this need by providing a more memory-efficient static array initialization. The invention of the '520 patent recognizes bytecode instructions that perform static array initialization and replaces the lengthy sequence of bytecode instructions with an instruction for performing the static array initialization. The replacement "sav[es] a significant amount of memory" (*id.* at 3:4-7), and thus solved the long-felt need to reduce memory footprint during initialization of arrays.

2. The '520 Patent Led to Commercial Success

461. I understand that testimony at trial will show customer demand for devices that use memory efficiently and thus can store more applications or data. The '520 patent has helped Android minimize its memory usage and thus helped it become commercially successful.

462. I have read Professor Mitchell's Opening Patent Infringement Report, Section IX entitled "6,061,520 (Play Execution)," and understand that the evidence at trial will show that Android's dx tool implements the '520 patent. Specifically, I understand that the evidence at trial will show that the dx tool processes each <clinit> method, including those that initialize static arrays.

463. Google has presented and marketed how efficiently the dx tool initializes arrays. Google explained the problem with conventional static array initialization: "sometimes you really need to have just a big array of data and if you've ever looked at what something like this looks like ... in a .class file, it's not pretty. So that, this really is the code that's needed to initialize that previous array. It's 44 bytes of instructions, another 35 bytes in the constant pool, and it's 4 instruction dispatches just to initialize each element. And each time you add another element to say ... an int array, it's 11 bytes of code and constant combined and another ... 4 instruction dispatches." (Google I/O Android Video on "Dalvik Virtual Machine Internals" by Dan Bornstein (2008), *available at* http://developer.android.com/videos/index.html#v=ptjedOZEXPM.) Google explained the benefit of using the technique of the '520 patent: "... we're only using 46 bytes for ... this example and as you add elements to that int array, it's just 4 more bytes per element, which is exactly the data represented. And we only have to interpret one opcode to do that entire initialization... that's fill-array-data. . . . And this is both a speed and a space efficiency win. Measured on our system libraries it saves us something like a 100K." (*Id.*) Based on Professor Mitchell's Opening Patent Infringement Report, I understand that the fill-array-data opcode refers to the claimed "instruction requesting the static initialization of the array", for example, of the '520 patent.

464. Furthermore, I have read Professor Mitchell's Opening Expert Report, Section IV entitled "Android's Success Is Due to the Claimed Inventions" and understand that Dr. Mitchell, together with Oracle Java engineer Noel Poore, have conducted benchmark testing analysis, which has confirmed that the performance of Android would be poor without the benefit of using the '520 patent. The performance benchmark testing compared the size in bytes of .dex files both with and without the static array initialization optimization of the claimed invention of the '520 patent. I understand that the benchmarking testing results showed that all .dex files were larger without the static array initialization optimization. I also understand that the benchmarking testing results showed as much as 57% memory savings for an array of size 100 of primitive data type int.

465. I understand that testimony at trial will show customer demand for devices that use memory efficiently and thus can store more applications or data. Based on the benchmark analysis, I conclude that the capacity of Android to load large numbers of applications would be reduced if it did not implement the claimed invention of the '520 patent, and thus would be a less attractive platform. Therefore, I understand that there is a nexus between the claimed invention of the '520 patent and Android's commercial success.

466. For at least the above reasons, it is my opinion that secondary considerations demonstrate the non-obviousness of the '520 patent.

E. '720 patent

1. The '720 Patent Solved a Long-Felt Need

467. Since the implementation of Java virtual machines, there has been a need among system developers to have efficient use of memory among multiple virtual machine processes, while providing a robust environment for executing the multiple virtual machine processes concurrently. While Java virtual machines are attractive for multi-process systems, developers were very much aware of the disadvantages associated with startup time and memory management of these virtual machines. For example, each virtual machine had a distinct address

space physically separate from that of the other virtual machines, and so each virtual machine consumed vital system resources. Additionally, initialization costs were repeated for each virtual machine. A later approach, implemented in an attempt to address memory management, involved the virtual machines sharing some memory between the processes, while maintaining individual memory spaces. This approach had its own drawbacks. Initialization costs were still repeated for each virtual machine and the individual memory spaces were not always necessary. Memory usage improved, but was still inefficient. This approach was not practical for smaller devices having limited memory resources.

468. The ’720 patent satisfied this need with a new approach to virtual machine memory management and startup that used process cloning with copy-on-write technology to share memory between a master virtual machine and a cloned virtual machine, until the cloned virtual machine needed to modify the shared memory. The ’720 patent’s approach is well-suited to smaller devices having limited memory resources because (a) it shares common libraries between processes, (b) it does not have to repeat initialization costs for each cloned virtual machine, and (c) it shares memory between processes by default. It also reduces startup time by cloning a prewarmed master virtual machine to create a child virtual machine.

2. The ’720 Patent Led to Commercial Success

469. I understand that testimony at trial will show customer demand for devices that use memory efficiently and can run multiple applications. The ’720 patent has helped Android minimize its memory usage and thus helped it become commercially successful.

470. I have read Professor Mitchell’s Opening patent Infringement Report, Section IV entitled “Android’s Success Is Due to the Claimed Inventions” and understand that Dr. Mitchell, together with Oracle Java engineers, Erez Landau and Seeon Birger, have conducted benchmark testing and analysis, which has confirmed that the performance of Android would be poor without the benefit of using the ’720 patent. I further understand that the performance benchmark testing results show that Android’s use of the ’720 patent results in as much as 40%

memory savings for Android, memory savings of 2MB per each additional running application, and also a 0.10 second per application launch time savings. Professor Mitchell describes that disabling the '720 functionalities in Android increases memory consumption by 70%.

471. Google has given presentations on the Zygote process to software developers. Google explained: "[the]VM process that initializes a Dalvik VM and preloads a lot of these libraries . . . . It uses copy-on-write to maximize re-use and minimize footprint so that data structures are shared and it won't do a full copy unless some of those data structures are to be modified." (Google I/O Video on "Anatomy and Physiology of an Android" by Patrick Brady (2008), *available at* http://developer.android.com/videos/index.html#v=G-36noTCaiA.) Based on Professor Mitchell's Opening Patent Infringement Report, I understand that the excerpt above refers to Zygote and the '720 patent. The fact that Google presented Zygote's ability to software developers indicates that the ability to "maximize re-use and minimize footprint" is significant.

472. Google has stated: "Application switching on a mobile device is extremely critical; we target significantly less than 1 second to launch a new application." (*Multitasking the Android Way*, *available at* http://developer.android.com/resources/articles/multitasking-android-way.html.) I find that the saving of 0.10 seconds that Zygote provides (as determined by Prof. Mitchell) is a substantial part of "significantly less than 1 second."

473. I understand that testimony at trial will show customer demand for devices that use memory efficiently and devices that can quickly startup applications. Based on the benchmark analysis, I conclude that without the substantial memory savings and application startup time savings provided by the claimed invention of the '720 patent, the capacity of Android to run large numbers of simultaneous applications, and the responsiveness of doing so, would be substantially diminished. Therefore, I understand that there is a nexus between the claimed invention of the '720 patent and Android's commercial success.

possible subsequent results of an action by a method." (*Id.* ¶ 37.) In my opinion, the fine-grained security model described in the '447 and '476 patents provides a more flexible model and offers greater protection from harmful actions.

2. The '447 and '476 Patents Led to Commercial Success

482. I understand that Sun Microsystems implemented and Oracle continues to implement the security model described in the '447 and '476 patents in Java. Li Gong's book entitled "Inside Java™ 2 Platform Security" explains the security model in detail:

> [E]ach class is associated—when it is defined—with an instance of ProtectionDomain. A ProtectionDomain is a convenience class for grouping . . . static permissions . . . . [A] resource access is granted if every ProtectionDomain in the current execution context has been granted the permission required for that access, that is, if the code and Principals specified by each ProtectionDomain are granted the permission. (Li Gong et al., *Inside Java™ 2 Platform Security, Second Edition* 58 (2003) (OAGOOGLE0000106419).)

483. The security model is implemented across multiple Java classes, including java.security.ProtectionDomain, and java.security.AccessController. (*See id.* at ch.5-6.)

484. Sun Microsystems first introduced the new security model in JDK 1.2 in December 1998. (Li Gong, *Java Security: A Ten Year Retrospective*, 2, *available at* http://www.acsac.org/2009/program/keynotes/gong.pdf (OAGOOGLE0100359495) ("JDK 1.2 introduced the fine-grained access control model, which continued essentially unchanged till this day.").) Even though it has been over a decade since Sun Microsystems introduced the fine-grained security model, "the overall architecture had in fact been carried over to both the enterprise Java and mobile Java platforms." (*Id.* at 1.) In fact, Java SE 5.0 and 6.0, for example, still include the java.security package that implements the fine-grained security model. (*See, e.g.*, Class ProtectionDomain for Java™ 2 Platform Standard Ed. 5.0, *available at* http://download.oracle.com/javase/1.5.0/docs/api/java/security/ProtectionDomain.html; Class ProtectionDomain for Java™ Platform Standard Ed. 6, *available at* http://download.oracle.com/javase/6/docs/api/java/security/ProtectionDomain.html.)

485. I understand that testimony at trial will show customer demand for devices with increased security protection. In my opinion, the functionalities of the '447 and '476 patents have contributed to the widespread adoption of the fine-grained security model in Java products. Therefore, I understand that there is a nexus between the claimed inventions of the '447 and '476 patents and Java's commercial success.

3. The '447 and '476 Patents Have Received Industry Praise

486. Java's fine-grained security model, which implements the functionalities of the '447 and '476 patents, has received praise from the industry. When Sun Microsystems released JDK 1.2, the industry praised the fine-grained security model for solving the earlier problems with Java's sandbox approach. (*See, e.g.*, *Web Based Programming Tutorials*, *available at* http://www.webbasedprogramming.com/Java-1.2-Unleashed/ch03.htm ("The security model implemented by the Java sandbox has been strengthened and at the same time made more flexible from JDK 1.0 to JDK 1.2."); Monica Pawlan, *JDK 1.2: New Features and Functionality* (May 1998), *available at* http://www.pawlan.com/monica/articles/jdk1.2features/ ("JDK 1.2 introduces a strong security model . . . . The JDK 1.2 security policy is easy to configure, provides fine-grained access control, and applies to all Java applets and applications."); Larry Koved et al., *The Evolution of Java Security* (May 24, 2003), *available at* http://www.developertutorials.com/tutorials/java/evolution-java-security-050524-1448/ ("The new JDK 1.2 permission model is much more flexible and even permits application-defined resources to be added to the access control system.").)

4. Google Copied the Claimed Inventions of the '447 and '476 Patents

487. I understand that the evidence at trial will show that, in developing its competing Android software, Google copied the desirable functionalities of Java's fine-grained security model claimed in the '447 and '476 patents. (*See, e.g.*, Android Developers Package Reference (listing Android APIs including the java.security package), *available at* http://developer.android.com/reference/java/security/package-summary.html.) The Android

Dated: August 25, 2011

DR. BENJAMIN F. GOLDBERG